**LAML教程**

# MORE & LAML 概述

LAML引擎脚本语言

LEWA Application Markup Language for MORE （LEWA MORE引擎应用标记语言）

MORE渲染引擎

Markup Oriented Rendering Engine （基于标记语言的渲染引擎）

概述

最初用于百变锁屏，使用xml用特定的语法描述锁屏界面。后来不断增强功能，逐步演化成一套接近通用的界面描述语言和图形渲染引擎，在一定需求下可用于开发风格多变的用户界面。可方便地通过更换皮肤改变界面风格、动画甚至交互方式。

LAML语言和Android的界面描述xml 类似. 所不同的是Android描述的是静态界面，对界面元素的更改依赖java代码。LAML描述的是静态界面+动态属性，UI在时间线上按一定的帧率不断刷新，UI显示根据元素属性的变量表达式的计算结果实时更新。LAML语言和运行时引擎已经从锁屏中独立出来作为LEWA内置的通用框架，除了显示时间日期等，还支持查询标准Content Provider来获取各种信息如天气。显示图片文本等各种元素，各种动画，滑动点击等界面交互控件，适于实现展示信息或有简单交互操作的界面。比如锁屏、动态图标。

框架支持动态帧率，不必按照固定帧率不停渲染，在没有动画和更新的时侯停止渲染，此时仅占用极少资源，对于缓慢变化的动画使用低帧率渲染，高动态的动画开始后立即调整到高帧率全速渲染。全速渲染时全屏帧率基本可以达到60帧。合理使用可以既炫酷又不费电。

**­LAML在百变锁屏中的应用**

百变锁屏在主题包里的lockscreen/advance目录下，manifest.xml文件是描述脚本

<Lockscreen frameRate="" displayDesktop="" showSysWallpaper="" screenWidth="720">

<Image />

<Image />

<Unlocker/>

<Unlocker/>

<Time/>

<DateTime/>

…

<Text/>

</Lockscreen>

* frameRate： 指定帧率，如果动画缓慢，可以指定小一点的值，省电。默认为30。
* displayDesktop: 默认为false，透视到桌面功能，如果没有锁屏壁纸或者锁屏壁纸可以被移开或透明时可以看到桌面launcher或者是锁屏前的应用程序。可以完美实现WP7，Meego，阿里云等的解锁界面。
* showSysWallpaper: 默认为false,是否在锁屏界面显示桌面壁纸。开启后如果没有指定锁屏壁纸<Wallpaper/>,会将桌面壁纸作为锁屏的壁纸。
* screenWidth： 设定屏幕宽度标准。如果指定为720,锁屏中所有元素的位置都按720p的布局编写，480p的手机会自动进行缩放。

**LAML在百变壁纸及动态图标中的应用**

动态图标

动态图标在主题包icons\fancy\_icons\目录下，每个动态图标是一个文件夹，文件夹的名字是对应的app包名。例如日历的动态图标是一个叫"com.android.calendar"的文件夹，里面包含manifest.xml描述文件

<Icon version="1" frameRate="" width="136" height="136" screenWidth="720" useVariableUpdater="" hideApplicationMessage="">

<Var/>

<Image/>

<Text/>

<DateTime/>

…

</Icon>

hideApplicationMessage: 默认false,屏蔽右上角的通知标志。

useVariableUpdater： 指定需要哪些系统变量，目前包括电量和时间，而且可以指定时间的更新周期，如果时间小工具每秒更新一次，需要指定更新周期为1秒，否则默认会每分钟更新一 次。如果小工具或动态图标只显示日期，则每天更新一次，可以指定更新周期为1天。尽量选择较长的更新周期，以节约系统资源。

目前支持的tag： Battery, DateTime.Day, DateTime.Hour,DateTime.Minute,DateTime.Second

如果某个小工具不需要电量或时间变量，则需要使用useVariableUpdater="none"，指定不做时间变量等更新。

如果某个小工具显示系统电量并且显示根据时间每小时更换背景图片: useVariableUpdater="Battery,DateTime.Hour"

如果需要每秒进行多次刷新，可以直接使用frameRate

**壁纸元素**

Wallpaper元素引用系统设置的壁纸，除了不能指定图片源外其他和Image元素相同，可以有动画和其他属性控制。如果没有此元素则不显示壁纸。可以有多个。 例子： <Wallpaper/>

**变量**

<Var>变量

<Var name="" expression="" type="" const="" threshold="" persist=""/>

* name 变量名
* expression 变量对应的表达式或常量 注意：如果定义字符串常量需要多一套单引号: expression="'my string'"
* type= number/string 定义数值变量或字符串变量 默认：number
* const =true 变量只会在初始化时计算一次，以后不会重新计算，可以提高效率。如果变量值会在锁屏运行期间改变，const设为false 默认：false
* threshold 阈值触发，当变量值的变化超过设定的阈值时，可以触发一些命令。例如：

<Var name="time3" expression="#minute%10" threshold="1">

<Trigger>

<Command target="time3\_anim.visibility" value="true"/>

<Command target="time3\_anim.animation" value="play"/>

<Command target="time3\_anim\_new.visibility" value="false"/>

<Command target="time\_3\_anim\_new.visiblity" value="true" delay="400"/>

<Command target="time\_3\_anim\_new.animation" value="play" delay="400"/>

</Trigger>

</Var>

上述代码表示#time3的值每变化1，就会执行<Trigger/>里面的所有命令

* persist 默认false，变量持久化。指定为ture后，如果没有重新给定该变量其他的值，那么这个值会一直保存，无论解锁后重新锁定或者重新应用主题都不会还原

**变量数组**

<VarArray>变量数组

<VarArray type="string">

<Vars>

<Var name="date\_format1" index="#time/2000%5"/>

<Var name="date\_format2" index="2" const="true"/>

...

</Vars>

<Items>

<Item expression="ifelse(isnull(@date\_format),'MMMMd日 EEEE', @date\_format)"/>

<Item value="EEEE"/>

...

</Items>

</VarArray>

* Item 数组元素定义
* expression 元素对应的表达式
* value 如果元素是常量，则用value指定常量值

// 定义了一个每隔两秒轮换一次日期显示格式的DateTime元素.

// 数组元素可以用来根据数字变量值显示不同的字符串

<!-- if const is true, the variable will only evaluate once at the initial time, -->

<Var name="show\_date" expression="ifelse(isnull(#show\_date),1,#show\_date )" type="number" const="true"/>

<Var name="text\_size\_date" expression="ifelse(isnull(#text\_size\_date),18,#tex t\_size\_date)" type="number" const="true"/>

<VarArray type="string">

<Vars>

<Var name="date\_format1" index="#time/2000%5"/>

<Var name="date\_format2" index="2" const="true"/>

</Vars>

<Items>

<Item expression="ifelse(isnull(@date\_format),'MMMMd日 EEEE', @date\_format)"/>

<Item value="EEEE"/>

<Item value="yyyy年"/>

<Item value="MMMM"/>

<Item value="d日"/>

</Items>

</VarArray>

// 显示英文月份

<VarArray type="string">

<Vars>

<Var name="month\_str" index="#month"/>

</Vars>

<Items>

<Item value="January"/>

<Item value="February"/>

...

<Item value="November"/>

<Item value="December"/>

</Items>

</VarArray>

<Text text="@month\_str" color="#FFFFFFFF" size="20" ... />

**全局变量**

unlocker:

* unlocker.move\_x 解锁部件在x方向的偏移
* unlocker.move\_y 解锁部件在y方向的偏移
* unlocker.move\_dist 解锁部件移动的距离
* unlocker.state 解锁部件的状态： 0 normal, 1 pressed, 2 reached

touch：

* touch\_x 当前触摸点的x坐标
* touch\_y
* touch\_begin\_x 按下屏幕时的初始x坐标
* touch\_begin\_y
* touch\_begin\_time 按下屏幕时的时间

time:

* time 当前时间，long
* time\_sys 系统时间毫秒数
* year 年份
* month 月份（取值范围是0~11，0表示一月，1表示二月，以此类推）
* date 日期
* day\_of\_week 星期（1表示星期日，2表示星期一，等等）
* hour12 当前小时（12小时制）
* hour24 24小时制
* minute 分钟
* second 秒
* ampm 上下午 0:am 1:pm
* time\_format 0:12h 1:24h

农历

* year\_lunar 农历年份
* year\_lunar1864 用来计算天干地支
* month\_lunar 农历月份 从1开始计
* month\_lunar\_leap 0/1 是否润月
* date\_lunar 农历日期 从1开始计

battery:

* battery\_level 当前电量，1~100
* battery\_state 0正常 1充电 2电量低 3已充满

screen:

* screen\_width 屏幕宽度
* screen\_height 屏幕高度
* raw\_screen\_width 物理宽度（这个表示当前设备的屏幕分辨率，不受根节点screenWidth的影响）
* raw\_screen\_height 物理高度

image:

* image.actual\_x 图片实际x坐标
* image.actual\_y
* image.actual\_w 图片实际宽度
* image.actual\_h
* image.bmp\_width 图片文件的宽度（不受裁切、缩放的影响，只根据src找到指定的图片，并检测该文件的宽度）
* image.bmp\_height

music:

* music\_control.music\_state 0暂停 1播放

others:

* sms\_unread\_count 未读短信数
* call\_missed\_count 未接电话数
* @next\_alarm\_time 下一个闹钟时间
* intercept\_sys\_touch 是否截获以后的触摸事件，避免被其他View捕获，例如在自由桌面widget中可以防止在widget上进行触摸操作时桌面滚动和进入编辑模式
* frame\_rate 当前屏幕帧率
* volume\_type

0 通话音量 1 系统音量 2 电话铃声，短信铃声 3 音乐播放器音量 4 闹钟音量 5 通知音量 6 连接蓝牙时的通话音量 7 在某些国家强制的系统音量 8 DTMF音量 9 TTS音量 10 FM音量

* ring\_mode 0 silent, 1 vibrate, 2 normal
* wifi\_state 0 disabled, 1 enabled, 2 problem, 3 half(intermediate)
* bluetooth\_state 0 off, 1 on, 2 half
* data\_state 0 off, 1 on
* usb\_mode 0 disconnected, 1 charge only, 2 storage, 3 intermediate

**表达式**

支持加减乘除取模括号和函数  
加减乘除取模 : + - \* / %  
支持函数:sin, cos, tan, asin, acos, atan, sinh, cosh, sqrt, abs, min, max 不解释

* len(数字) 给定数字位数 len(1234)=4
* digit(数字, 第几位) 取给定数字的第几位 digit(1234, 2) = 3
* substr(原字符串，字串开始位置，字串长度) substr('今天真热',1,2) = '天真'
* round() 四舍五入取整 int()向下取整
* eq(x, y) // x==y ? 1 : 0
* ne(x, y) // x!=y ? 1 : 0
* ge(x, y) // x>=y ? 1 : 0
* gt(x, y) // x>y ? 1 : 0
* le(x, y) // x<=y ? 1 : 0
* lt(x, y) // x isnull(x) // x==null ? 1 : 0 // 是否表达式包含空变量，通常用于判断绑定的变量是否查到数据
* not(x) x>0 ? 0 : 1
* ifelse(x, y, z) // x>0 ? y : z
* ifelse(x1, y1, x2, y2, ... , z)
* if x1>0 return y1; else if x2>0 return y2; ... ; else return z
* +可以拼接字符串 ， 一些函数也可以支持字符串，isnull可以判断字符串变量是否为空，ifelse可以选择字符串，字符串比较函数eqs(@string1, @string2)

元素可见性支持表达式 visibility=“表达式” <=0 不可见 >0可见  
Image, Text, Time的属性  
align= left, center, right 坐标点水平对齐方式  
alignV= top, center, bottom 坐标点垂直对齐方式

**文本**

显示指定格式的文字，坐标属性支持变量

* color： 文字颜色，#FFFFFF
* size: 文字大小
* format： 如果需要在文字中显示变量数字，需要指定格式, 用%d 指定数字位置
* paras: 如果指定了format, 需要在paras里指定%d对应的变量表达式, 可以有多个变量表达式用"," 隔开
* align： left, center, right，文字坐标的对齐方式
* textExp: 文字表达式，可以直接调用变量等如想要输出“现在时间是9点”，可以写成textExp="'现在时间是'+#hour12+'点'"
* width 文字宽度,当文字超过指定宽度时会被切掉。如果指定了多行显示，则会折行显示。如果指定了文字滚动，则会在指定的位置滚动显示文字
* marqueeSpeed 文字滚动速度，配合上面的宽度使用
* marqueeGap 滚动间隔。当文字显示完后再次出现的间隔，默认为四个汉字的宽度
* rotation（X,Y,Z） 旋转角度，旧的angle也可以使用，不过建议用rotation，因为比较高端大气国际化
* multiLine ture/false 是否支持多行显示，默认false
* spacingMult 行距倍数 默认1
* spacingAdd 行距增加量 默认0
* text.text\_width 某行文本的宽度，可以用来排版

<Text name="tt" .../>

#tt.text\_width

实例用法：

显示下一个闹钟时间

<Text text="@next\_alarm\_time" .../>

<Text format="下一个闹钟:[%s] 电池:[%d%%]" paras="@next\_alarm\_time,#battery\_level"/>

textExp属性支持字符串表达式：

\*天气信息，如果温度为空显示-- 非空显示数值

<Text visibility="#hasweather" x="(#screen\_width/2)+180" y="#screen\_height-290" w="360" size="36" color="#ffffff" align="right" textExp="ifelse(isnull(#weather\_temperature),'--',#weather\_temperature) + '℃'"/>

**时间**

src表示时间图片的前缀，如下表示使用time\_0.png, time\_1.png, ... time\_9.png, time\_dot.png. 坐标属性支持变量表达式

space表示时间图片的间隙，我们可以使用这个功能来对时间图片进行排版，使用正值时图片间距变大，这时可以将图片切小，节省内存。对于有投影的图片，将space写成负值，可以使投影重叠以节省空间

**日期**

显示指定格式的日期  
format: 支持标准日期格式, 另外增加农历: NNNN

<DateTime x="" y="" color="" size="" format="">

<PositionAnimation/>

<DateTime>

* 九月:

M -> 9  
MM -> 09

* 7分钟:

m -> 7  
mm -> 07  
mmm -> 007  
mmmm -> 0007

* 1970年4月6日 3:23am

"yyyy/MM/dd h:mmaa" -> "1970/04/06 3:23am"  
"M月d日 h:mmaa" -> "4月6日 3:23am"

* 星期

"E" -> "周三"  
"EEEE" -> "星期三"  
"纪念日: M/d" -> "纪念日: 4/6"

* 24小时: "kk:mm" -> "13:34"

**图片**

图片部件用来在锁屏界面上显示一个图片，可以指定各种属性

<Image x="" y="" w="" h="" pivotX="" pivotY="" rotation="" src="" srcid="" alpha="" align="" antiAlias=""/>

* x,y 相对于屏幕左上角的坐标
* w,h 宽和高
* pivotX, pivotY 旋转中心
* rotation 旋转角度，一周360度
* src 图片名称
* srcid 图片序列后缀数字，一般用变量表示，可以根据变量显示不同的图片，如果src="pic.png" srcid="1" 则最后会显示图片 "pic\_1.png"
* alpha 透明度 0-255, 小于等于0不显示
* antiAlias true/false 抗锯齿，如果为true图片在变形旋转时不会有锯齿，但是速度会慢
* srcExp 图片源表达式
* srcFormat 图片源格式
* srcForamtExp 图片源格式表达式
* srcParas 图片源参数
* actual\_x actual\_y 元素实际位置变量，可以获得元素在动画时的实际位置
* align/alignV 对齐方式，上文已经介绍。

例子:

<Image x="0" y="#screen\_height-323" src="bottom\_bg.png"/>

<Image name="img" ... >

<PositionAnimation>

...

</PositionAnimation>

</Image>

//srcExp:图片源表达式

<Image x="210" y="-26" align="center" srcExp="'weather/weather\_' + #weather\_id + @icon\_suffix + '.png'"/>

**数字图片**

<ImageNumber number="" src="">

* number是要显示的数字表达式
* src是图片源的文件名，支持SourceAnimation

如果src="number.png" 则会使用 number\_0.png number\_1.png ... 图片文件来绘制数字。类似Time部件

**图片遮罩**

<Image x="444" y="#screen\_height-92" src="hs\_path\_light.png">

<PositionAnimation>

<Position x="-438" y="0" time="2000"/>

</PositionAnimation>

<Mask x="0" y="#screen\_height-92" src="hs\_path\_mask\_r.png" align="absolute"/>

</Image>

* x,y 坐标
* src 遮罩图片，不透明黑色部分表示不透明，其他部分为透明。
* align 坐标是相对于所属图片还是绝对位置，如果是相对，图片移动是遮罩会相应跟随移动。否则遮罩保持不动。

遮罩同样可以有源动画，旋转动画，和位置动画，属性支持变量表达式

<Mask x="" y="" src="" centerX="" centerY="" angle="" align="">

<SourcesAnimation/>

<RotationAnimation/>

<PositionAnimation/>

</Mask>

**元素动画**

所有元素都支持动画  
动画分为：图片源，位置，大小，旋转，透明度  
每种动画相互独立，各自循环播放，动画由若干关键帧组成，关键帧包括帧属性和时间，除图片源动画外，其它动画会根据当前时间找到相邻的两个关键帧，然后线 性插值计算当前的属性。 如果第一帧时间不从0开始，则默认时间为0的第一帧为图片原始属性,时间单位为毫秒。 位置动画中的位置是相对于图片自身的坐标。

<Image>

<PositionAnimation>

<Position x="10" y="20" time="100"/>

<Position x="100" y="200" time="1000"/>

</PositionAnimation>

</Image>

<RotationAnimation>

<Rotation angle="" time=""/>

<Rotation angle="" time=""/>

</RotationAnimation>

<SizeAnimation>

<Size w="" h="" time=""/>

<Size w="" h="" time=""/>

</SizeAnimation>

<!-- 0-255 -->

<AlphaAnimation>

<Alpha a="" time=""/>

<Alpha a="" time=""/>

</AlphaAnimation>

例子：位置动画表示1秒从屏幕最左端到最右端，停留1秒，透明度动画表示开始透明度为175，在从最左端到最右端过程中透明度不变，到达最右端后0.5秒渐变为不透明，然后0.5秒变为透明消失。 然后循环播放。

<Image x="0" y="#screen\_height-177" src="charging\_light.png" category="Charging">

<PositionAnimation>

<Position x="480" y="0" time="1000"/>

<Position x="480" y="0" time="2000"/>

</PositionAnimation>

<AlphaAnimation>

<Alpha a="175" time="0"/>

<Alpha a="175" time="1000"/>

<Alpha a="255" time="1500"/>

<Alpha a="0" time="2000"/>

</AlphaAnimation>

</Image>

注意：只有图片支持图片源动画  
图片源动画稍有不同，没有插值，x, y 可选，表示相对图片的位置， 当前的图片是在列表里的找到第一个大于当前时间的那个点指定的

<SourcesAnimation>

<Source x="" y="" src="pic1.png" time="100"/>

<Source x="" y="" src="pic2.png" time="1000"/>

</SourcesAnimation>

**Slider**

<Slider name="">

* name: 名字，用来做变量名
* bounceInitSpeed, bounceAccelation 回弹动画初始速度和加速度(距离单位为像素，时间单位为秒)，都支持表达式
* alwaysShow="true/false" 默认是false，当一个Slider可见时，其他Slider消失。
* 可以有任意个Slider部件。
* Trigger Slider的状态切换时，支持Trigger触发

Slider起始点：起始点中包含的元素都会跟随当前操作的拖动移动位置。

<StartPoint x="" y="" w="" h="" normalSound="" pressedSound="">

<!--正常状态，可以包含若干元素-->

<NormalState>

<Image/>

<Time/>

<DateTime/>

<Text/>

//从其他状态切换到NormalState时，触发该Trigger

<Trigger>

<Command target="test\_trigger.visibility" value="false"/>

</Trigger>

</NormalState>

<!--在点击区域按下，表示开始触发操作，拖动，隐藏其他状态元素，显示这些元素-->

<PressedState>

<Image/>

<Time/>

<DateTime/>

<Text/>

//从其他状态切换到PressedState时，触发该Trigger

<Trigger>

<Command target="test\_trigger.visibility" value="true"/>

</Trigger>

</PressedState>

<!--达到触发目标后，隐藏其他元素，显示这些元素-->

<ReachedState>

<Image/>

<Time/>

<DateTime/>

<Text/>

//从其他状态切换到ReachedState时，触发该Trigger -->

<Trigger>

<SoundCommand sound="reached.mp3" volume="1"/>

</Trigger>

</ReachedState>

</StartPoint>

三个状态都可以不指定，可以使用变量来改变其他界面元素状态来表示触发过程。  
  
Slider目标点：当起始点的x,y点落入EndPoint目标区域矩形时，达到触发位置，此时松开手指即可以产生触发，可以有多个触发目的。

<EndPoint x="" y="" w="" h="" reachedSound="">

<!--Slider执行的动作，可以指定不同的intent-->

< Intent action="" type="" category="" package="" class=""/>

<NormalState>

</NormalState>

<!-- show the target position if pressed trying to unlock-->

<PressedState>

<Image/>

</PressedState>

<!-- show specified image when reached target, if touch up then perform unlock-->

<ReachedState>

<Image/>

</ReachedState>

<!--Slider路径 tolerance：开始触发操作后移动时路径最大距离超过这个值则取消，回到正常状态。Path可以指定x,y坐标，默认为0，Position的坐标为相对坐标。-->

<Path x="" y="" tolerance="">

<Position x="" y="" />

<Position x="" y="" />

</Path>

</EndPoint>

例子:

<Slider name="slider">

<StartPoint x="31" y="#screen\_height-117" w="90" h="90">

<NormalState>

<Image x="31" y="#screen\_height-117" src="unlock\_button.png">

</Image>

</NormalState>

</StartPoint>

<EndPoint x="359" y="#screen\_height-117" w="90" h="90">

<PressedState>

<Image x="359" y="#screen\_height-117" src="unlock\_target.png">

</Image>

</PressedState>

<Path x="0" y="#screen\_height-117">

<Position x="31" y="0" />

<Position x="359" y="0" />

</Path>

</EndPoint>

</Slider>

* Slider的intent也可以为Trigger和Command，一个Trigger可以有多个Command。Trigger和Command请见[Trigger & Command](http://wiki.dev.xiaomi.com/index.php?title=MAML%E6%95%99%E7%A8%8B#Trigger_Command)

Unlocker继承Slider，以上用法只需把tag名的Slider改为Unlocker.

* 锁屏音效

StartPoint: normalSound, pressedSound ,指定在normal和press状态播放的音效  
EndPoint: reachedSound, 到达该endpoint后播放的音效

<Unlocker name="unlocker">

<StartPoint x="0" y="440" w="480" h="400" normalSound="sound\_normal.mp3" pressedSound="sound\_normal.mp3">

<EndPoint x="0" y="640" w="480" h="400" reachedSound="sound\_reached.mp3">

</Unlocker>

**按钮**

按钮元素可以用来接收点击，双击等事件，并可根据trigger的定义来控制界面上其他元素。按钮元素也可以将事件传递给界面上其他元素，来使其他元素响应用户在锁屏界面上的操作。

<Button name="" x="" y="" w="" h="" listener="">

所有元素增加name属性，按钮的控制对象通过name来指定。

* x, y, w, h 指定了按钮区域。
* listener 指定该按钮的事件要传给哪个元素。

一个Button可以有若干个trigger，trigger定义了按钮动作引发相关的操作，如控制某元素的属性

<Trigger action="" target="" property="" value=""/>

* action: 按钮动作： down (按下), up (抬起), double (双击)
* target: 控制目标名
* property: 属性名，目前支持：visibility 以后会增加其他属性控制。
* value: 属性值，目前针对boolean有: true, false, toggle

通常用button控制的目标有三类，分别是可见性控制，播放动画，以及执行命令，简单演示：

<Button>

<Triggers>

<Trigger action="down/up/double">

<Command target="test.visibility" value="true/false">

<Command target="test.animation" value="play">

<VariableCommand name="test1" expression="0">

</Trigger>

</Triggers>

</Button>

关于命令<Command/>在后面会有详细的介绍

Button元素的xml结构：

<Button>

<!--可以指定任意个trigger，也可以不指定-->

<Triggers>

<Trigger/>

<Trigger/>

</Triggers>

<!--按钮正常状态，可以包含任意界面元素，如Image Text等，可以不指定

注意： Button中 Image的坐标和Button自己的坐标是独立的，都是相对于Button的父元素。-->

<Normal>

<Image/>

<Text/>

...

</Normal>

<!--按钮按下状态，可以包含任意界面元素，如Image Text等，也可以不指定，如果指定了normal状态，则显示normal状态元素。-->

<Pressed>

<Image/>

<Text/>

...

</Pressed>

</Button>

**音乐播放器**

指定名称，从而使显示与否可以由另一个按钮控制，比如双击。 必须包含4个Button和1个Text，并且name分别为指定的名称。可以包含其他任意界面元素如Image等。

* autoShow true/false 如果在播放音乐时进入锁屏，是否自动显示MusicControl
* music\_state 播放状态 0停止 1播放
* 更新后音乐播放支持歌名和歌手名字分开显示，分别使用@music\_control.title和@music\_control.artist

<MusicControl name="music\_control" x="" y="">

<!--专辑封面，和普通Image一样除了图片是绑定到专辑封面的，可以指定大小，加Mask等-->

<Image name="music\_album\_cover"/>

<!--歌名和歌手显示文字-->

<Text name="music\_display"/>

<!--上一首按钮-->

<Button name="music\_prev"/>

<!--下一首按钮-->

<Button name="music\_next"/>

<!--播放按钮-->

<Button name="music\_play"/>

<!--暂停按钮-->

<Button name="music\_pause"/>

</MusicControl>

例子：双击控制音乐控制显示隐藏

<Button x="178" y="529" w="126" h="126">

<Triggers>

<Trigger action="double" target="music\_control" property="visibility" value="toggle"/>

</Triggers>

</Button>

<MusicControl x="0" y="300" name="music\_control" visibility="false">

<Image name="music\_album\_cover" x="240" y="40" alpha="150" align="center" alignV="center" />

<!--背景图-->

<Image src="music\_bg.png" x="40" y="0" alpha="100" />

<!--Text新增自动滚动功能，如果制定了宽度和滚动速度marqueeSpeed，在文字超过宽度的时候会自动滚动显示-->

<Text name="music\_display" x="240" y="10" w="300" marqueeSpeed="30" size="24" color="#FFFFFF" alpha="150" align="center"/>

<Button name="music\_prev" x="40" y ="40" w="108" h="84">

<Normal>

<Image src="music\_previous\_n.png" x="40" y="40" />

</Normal>

<Pressed>

<Image src="music\_previous\_p.png" x="40" y="40" />

</Pressed>

</Button>

<Button name="music\_next" x="332" y ="40" w="108" h="84">

<Normal>

<Image src="music\_next\_n.png" x="332" y="40" />

</Normal>

<Pressed>

<Image src="music\_next\_p.png" x="332" y="40" />

</Pressed>

</Button>

<Button name="music\_play" x="186" y ="40" w="108" h="84">

<Normal>

<Image src="music\_play\_n.png" x="186" y="40" />

</Normal>

<Pressed>

<Image src="music\_play\_p.png" x="186" y="40" />

</Pressed>

</Button>

<Button name="music\_pause" x="186" y ="40" w="108" h="84">

<Normal>

<Image src="music\_pause\_n.png" x="186" y="40" />

</Normal>

<Pressed>

<Image src="music\_pause\_p.png" x="186" y="40" />

</Pressed>

</Button>

</MusicControl>

**音量调节**

* volume\_level 现在音量
* volume\_level\_old 调节之前的音量 取值： 1-15 根据二者比较判断是增大还是减小
* volume\_type
  + 0 通话音量
  + 1 系统音量
  + 2 电话铃声，短信铃声
  + 3 音乐播放器音量
  + 4 闹钟音量
  + 5 通知音量
  + 6 连接蓝牙时的通话音量
  + 7 在某些国家强制的系统音量
  + 8 DTMF音量
  + 9 TTS音量
  + 10 FM音量

一般锁屏下只能调3(音量播放器音量)  
volume\_type>=0 表示正在调节音量，调节完毕后值为-1 可根据这个显示或隐藏音量显示

**Content Provider**

content provider支持where的格式化：  
where="" whereFormat="" whereParas=""  
content provider 支持依赖关系，即某个ContentProviderBinder查询结束后获取的变量作为下一个ContentProviderBinder查询的参数

<ContentProviderBinder name="name1" dependency="name2"> <ContentProviderBinder name="name2" >

name2查询结束后会触发name1的查询，name1的查询可以使用name2的变量。并且如果name2数据发生变化重新查询后，会触发name1的重新查询。

content provider提供了查询应用程序信息的通用接口，定义了新的xml代码来查询content provider，并查询到的信息绑定到变量上，用来显示第三方应用程序的信息，只要第三方应用提供相应的content provider。比如可以显示天气信息，邮件，待办事项，便签，等等，你也可以写一个应用程序查询任何你想要的信息并通过content provider来提供给锁屏使用。

<VariableBinders>

<ContentProviderBinder uri="content://sample/test" c uriFormat="" uriParas="" columns="col1,col2" where="" args="" order="" countName="count\_name">

<Variable name="variable\_name1" type="int" column="col1" row="0"/>

<Variable name="variable\_name2" type="string" column="col2" row="0"/>

</ContentProviderBinder>

</VariableBinders>

* VariableBinders: 定义各种变量绑定到的源，目前仅支持content provider。
* ContentProviderBinder: 定义一个content provider源和绑定到它上面的变量
  + uri: content provider uri, 指定选用哪个content provider
  + uriFormat: 如果uri需要添加变量，可以用格式化，需要和uriParas一起使用
  + uriParas: 同Text element 的格式
  + columns: 需要查询的列名，用逗号分隔
  + where: 查询条件，同 SQL
  + args: "where" 的参数.
  + order: 排序条件， 同 SQL
  + countName: 将查询结构数量绑定到该变量名
* Variable: 定一个绑定变量
  + name: 变量名
  + type: content provider中的数据类型： string/double/float/int/long
  + column: 变量绑定到的列的名称.
  + row: 变量绑定到的行数，默认为0.

<!--从谷歌资讯天气查询天气信息--><nowiki>

<VariableBinders>

<ContentProviderBinder uriFormat="content://com.google.android.apps.genie.geniewidget.weather/weather/current/%d" uriParas="#time\_sys" columns="iconResId,location,temperature,lowTemperature,highTemperature,description" countName="hasweather">

<Variable name="weather\_id" type="int" column="iconResId"/>

<Variable name="weather\_location" type="string" column="location"/>

<Variable name="weather\_temperature" type="int" column="temperature"/>

<Variable name="weather\_lowTemperature" type="int" column="lowTemperature"/>

<Variable name="weather\_highTemperature" type="int" column="highTemperature"/>

<Variable name="weather\_description" type="string" column="description"/>

</ContentProviderBinder>

</VariableBinders>

<nowiki><!-- 显示天气信息 -->

<Text visibility="#hasweather" x="240" y="60" w="360" size="24" color="#FFFFFF" alpha="200" align="center" format="%s  %s  %d℃" paras="@weather\_location,@weather\_description,#weather\_temperature" />

<Text visibility="#hasweather\*not(isnull(#weather\_lowTemperature))\*not(isnull(#weather\_highTemperature))" x="240" y="85" w="360" size="22" color="#FFFFFF" alpha="200" align="center" format="%d℃ / %d℃" paras="#weather\_lowTemperature,#weather\_highTemperature"/>

<Text visibility="#hasweather\*isnull(#weather\_lowTemperature)\*not(isnull(#weather\_highTemperature))" x="240" y="85" w="360" size="22" color="#FFFFFF" alpha="200" align="center" format="--℃ / %d℃" paras="#weather\_highTemperature"/>

<Text visibility="#hasweather\*not(isnull(#weather\_lowTemperature))\*isnull(#weather\_highTemperature)" x="240" y="85" w="360" size="22" color="#FFFFFF" alpha="200" align="center" format="%d℃ / --℃" paras="#weather\_lowTemperature" />

<Image src="weather.png" visibility="#hasweather" x="240" y="100" srcid="#weather\_id" align="center">

<PositionAnimation>

<Position x="-10" y="0" time="1000"/>

<Position x="10" y="0" time="3000"/>

<Position x="0" y="0" time="4000"/>

</PositionAnimation>

</Image>

天气id列表：

* weather\_chance\_of\_rain = 2130837536;
* weather\_chance\_snow = 2130837540;
* weather\_chance\_storm = 2130837544;
* weather\_clear = 2130837556;
* weather\_cloudy = 2130837560;
* weather\_flurries = 2130837564;
* weather\_fog = 2130837568;
* weather\_heavy\_rain = 2130837572;
* weather\_icy\_sleet = 2130837576;
* weather\_mist = 2130837583;
* weather\_partly\_cloudy = 2130837587;
* weather\_rain = 2130837600;
* weather\_snow\_rain = 2130837606;
* weather\_snow = 2130837608;
* weather\_sunny = 2130837612;
* weather\_thunderstorm = 2130837617;
* weather\_unknown = 2130837621;
* weather\_windy = 2130837626;

查询便签内容，显示第一条便签内容和修改时间

<VariableBinders>

<ContentProviderBinder uri="content://notes/note" columns="modified\_date,snippet" order="modified\_date DESC" countName="note\_count">

<Variable name="note\_date1" type="long" column="modified\_date" row="0"/>

<Variable name="note\_snippet1" type="string" column="snippet" row="0"/>

</ContentProviderBinder>

</VariableBinders>

<DateTime x="300" y="50" color="#ffffffff" visibility="#note\_count" value="#note\_date1" format="yyyy年MMMMdd日 hh:mm"/>

<Text x="5" y="180" w="450" color="#ffffffff" visibility="#note\_count" size="18" marqueeSpeed="60" text="@note\_snippet1"/>

查询未接来电的参数

uri ="content://call\_log/calls";

columns = "\_id,number"

where = "type=3 AND new=1";

**Trigger & Command**

* Command

基础命令，可以通过对象名和对象的属性来控制界面里的其他元素。通常控制的是元素的可见性（visibility）和动画播放（animation）。

<Command target="iamge1.visibility" value="false" condition="" delay="" delayCondition=""/>

<Command target="iamge1.animation" value="play" condition="" delay="" delayCondition=""/>

1. iamge1是被控制对象的名字，“.”后面跟的是属性（目前支持visibility和animation）。
2. 控制可见性的时候，value里面写true或者false，控制动画的时候，value写play（目前不支持让动画暂停）
3. condition是条件判断，支持表达式。当condition里的条件判断为真时，执行命令；为假时，不执行。
4. delay是延迟，以毫秒记。读取该命令后延迟一段时间再执行
5. delayCondition是延时判断，在delay的时间之后再进行判断。

例子：

<Button x="0" y="0" h="100" w="100" >

<Triggers>

<Trigger action="down">

// second<40时，延时2000ms执行

<VariableCommand name="delayCond0" expression="#delayCond0+5" condition="lt(#second,40)" delay="2000" />

// 延时6000ms后，如果second<40，执行

<VariableCommand name="delayCond1" expression="#delayCond1+5" delayCondition="lt(#second,40)" delay="6000" />

// second<40时，延时4000ms，此时如果second<40，执行

<VariableCommand name="delayCond2" expression="#delayCond2+5" condition="lt(#second,40)" delayCondition="lt(#second,40)" delay="4000" />

</Trigger>

</Triggers>

</Button>

* VariableCommand

变量命令，用来控制变量（Var）的值。包括name和expression两个特殊属性，condition、delay、delayCondition的用法与Command一致

//将屏幕的宽度值赋值给w

<VariableCommand name="w" expression="#screen\_width"/>

* SoundCommand

声音命令，可以用来播放音频文件

<SoundCommand sound="" volume="" loop="" keepCur="" />

1. sound: 声音文件名
2. volume: 声音大小，0~1的一个浮点数
3. loop: 是否循环播放，true/false，默认是false.
4. keepCur: 播放此音频时，是否保持当前正在播放的声音，true/false，默认false.

例子：

// 循环播放reached.mp3，同时不停掉正在播放的其他声音

<SoundCommand sound="reached.mp3" volume="1" loop="true" keepCur="true" />

* ExternCommand

通用命令，用来向外部程序发送命令

<ExternCommand command="命令名" numPara="参数1,数字表达式" strPara="参数2,字符串表达式"/>

目前可以使用的一种命令是解锁命令(仅在锁屏中有效)

<ExternCommand command="unlock"/>

用这个命令可以实现解锁操作，无需使用<Unlocker/>

* ExternalCommand

与ExternCommand相对，是用来接收外部命令的命令，典型的用法：在锁屏中，通常用来接收开屏/关屏命令，从而执行一些命令；在桌面插件中，用来检测切屏从而执行命令

resume表示开屏时执行的命令，pause表示关屏时执行的命令

<ExternalCommands>

<Trigger action="resume">

<Command target="target.animation" value="play"/>

</Trigger>

<Trigger action="pause">

<VariableCommand name="pause\_time" expression="#time\_sys"/>

</Trigger>

</ExternalCommands>

桌面插件切屏时使用的示例

<ExternalCommands>

<Trigger action="resume">

<Command target="\_\_root.animation" value="play"/>

</Trigger>

<Trigger action="pause">

<Command target="\_\_root.animation" value="play"/>

</Trigger>

</ExternalCommands>

用按钮实现开关

<!--每个Button中可以有若干个Command-->

<Button x="10" y="10" w="100" h="100">

<Triggers>

<Trigger action="up">

<Command target="Wifi" value="toggle"/>

</Trigger>

</Triggers>

<Pressed>

<Image x="10" y="10" src="toggle\_btn\_pressed.png"/>

</Pressed>

<Normal>

<Image x="10" y="10" src="toggle\_btn\_normal.png"/>

</Normal>

</Button>

* 蓝牙 value="on" value="off"

<Command target="Bluetooth" value="toggle"/>

* 数据

<Command target="Data" value="toggle"/>

<!--仅切换到开，用另一个button切换到关-->

<Command target="Data" value="on"/>

<Command target="Data" value="off"/>

* 铃音/静音/震动

<!-- value="normal,silent,vibrate" value="silent" value="normal" <br>

在逗号分隔的状态键切换，可以只指定一个状态，多个button各自用不同的状态value值，可以快速切换到需要的状态 --> 三种状态切换

<Command target="RingMode" value="toggle"/>

下面和上面同样效果

<Command target="RingMode" value="normal,silent,vibrate"/>

仅正常和静音切换

<Command target="RingMode" value="normal,silent"/>

仅正常和震动切换

<Command target="RingMode" value="normal,vibrate"/>

仅切换到静音，用其他button切换到其他状态

<Command target="RingMode" value="silent"/>

* USB存储 toggle/on/off

<Command target="UsbStorage" value="toggle"/>

<Command target="UsbStorage" value="on"/>

<Command target="UsbStorage" value="off"/>

* Wifi toggle/on/off

<Command target="Wifi" value="toggle"/>

用滑动方式实现开关

<Slider name="switch\_usb\_mode" bounceInitSpeed="500" bounceAcceleration="1500" alwaysShow="true" visibility="#usb\_mode" >

<StartPoint x="0" y="0" w="100" h="100">

<NormalState>

</NormalState>

</StartPoint>

<EndPoint x="340" y="0" w="100" h="100">

<Command target="UsbStorage" value="toggle"/>

<Path >

<Position x="0" y="0" />

<Position x="340" y="0" />

</Path>

</EndPoint>

</Slider>

* 状态变量，仅在添加相应command后才有效
  + #ring\_mode: 0 silent, 1 vibrate, 2 normal
  + #wifi\_state: 0 disabled, 1 enabled, 2 problem, 3 half(intermediate)
  + #bluetooth\_state: 0 off, 1 on, 2 half
  + #data\_state: 0 off, 1 on
  + #usb\_mode: 0 disconnected, 1 charge only, 2 storage, 3 intermediate

Trigger的Command支持变量赋值

<VariableCommand name="test" expression="#test+1" /> name：变量名，expression：赋值表达式

Trigger的Command支持动画播放控制

<Button x="0" y="#screenview\_y" w="#screen\_width" h="#screenview\_h">

<Triggers>

<Trigger action="up">

<VariableCommand name="gesture\_valid" expression="ge(abs(#gesture\_offset),80)" />

<VariableCommand name="ani\_begin\_x\_add" expression="ifelse(lt(#current\_screen,#screenview\_count-1),#gesture\_offset+#screenview\_w,#gesture\_offset)"/>

<VariableCommand name="ani\_begin\_x\_dec" expression="ifelse(gt(#current\_screen,0),#gesture\_offset-#screenview\_w,#gesture\_offset)"/>

<VariableCommand name="ani\_begin\_x" expression="ifelse(#gesture\_valid,ifelse(#gesture\_offset,#ani\_begin\_x\_dec,#ani\_begin\_x\_add),#gesture\_offset)" />

<VariableCommand name="touch\_x" expression="#touch\_begin\_x"/>

<Command target="ani\_offset\_x.animation" value="play"/>

</Trigger>

</Triggers>

</Button>

<Var name="ani\_offset\_x">

<VariableAnimation>

<AniFrame value="#ani\_begin\_x" time="0"/>

<AniFrame value="0" time="300"/>

<AniFrame value="0" time="100000000000000"/>

</VariableAnimation>

</Var>

**动态帧率**

* 简单用法

除了原来的frameRate属性控制普通状态下的帧率外，还支持分别设置充电、电量低、充满电状态下的帧率，以方便的控制这些状态下的帧率，达到 省电的目标。比如普通状态下没有动画，帧率可以设为0，充电状态下有动画，帧率可以设为30，以显示平滑的动画，这样普通状态下就可以非常省电。  
例如：

frameRateCharging="30" frameRateBatteryLow="20" frameRateBatteryFull="1"

* 高级用法

为了在xml描述语言中根据时间线指定不同帧率，新增一种element，叫做FramerateController，此元素和其他界面元素一样可以被添加到各个容器中，此元素包含一个帧率控制时间线，指定在一定的时间使用一定的帧率。

<FramerateController visibility="#music\_control.visibility" loop="true">

<ControlPoint time="0" frameRate="20"/>

<ControlPoint time="1000" frameRate="0"/>

<ControlPoint time="3000"/>

</FramerateController>

此元素会控制它所在容器下的时间线的帧率，即，所有该容器下的元素的动画都会与此元素指定的帧率变化同步。

**传感器调用**

新增<SensorBinder type="">可以将传感器数据绑定到变量上，实现重力感应，方向感应，加速度感应，气压感应（海拔高度）

<VariableBinders>

//方向传感器：

type="orientation"

index=0： 方位角，0~359，0=北，90=东，180=南，270=西

index=1： 俯仰角，-180～180，z轴转向y轴为正方向

index=2： 滚转角，-90～90，x轴转向z轴为正方向

<SensorBinder type="orientation">

<Variable name="" index="0"/>

<Variable name="" index="1"/>

<Variable name="" index="2"/>

</SensorBinder>

//线性加速度

type="linear\_acceleration"

index=0： x方向的线性加速度

index=1： y方向的线性加速度

index=2： z方向的线性加速度

线性加速度是去掉了重力加速度影响的：加速度 = 线性加速度 + 重力加速度

<SensorBinder type="linear\_acceleration">

<Variable name="linear\_acceleration\_x" index="0"/>

<Variable name="linear\_acceleration\_y" index="1"/>

<Variable name="linear\_acceleration\_z" index="2"/>

</SensorBinder>

//加速度

type="accelerometer"

index=0： x方向的加速度

index=1： y方向的加速度

index=2： z方向的加速度

<SensorBinder type="accelerometer">

<Variable name="" index="0"/>

<Variable name="" index="1"/>

<Variable name="" index="2"/>

</SensorBinder>

//重力传感器

type="gravity"

index=0： x方向的重力加速度

index=1： y方向的重力加速度

index=2： z方向的重力加速度

<SensorBinder type="gravity">

<Variable name="" index="0"/>

<Variable name="" index="1"/>

<Variable name=" index="2"/>

</SensorBinder>

//气压传感器

type="pressure"

index=0： 气压值，单位hPa。海平面的平均气压是1013.25hPa，可以根据气压值估计海拔高度。

<SensorBinder type="pressure">

<Variable name="" index="0"/>

</SensorBinder>

</VariableBinders>